

EV316935344
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Software Build Extensibility

Inventor(s):
David S. Ebbo
Ting-Hao Yang

ATTORNEY'S DOCKET NO. MS1-1668US

Software Build Extensibility

TECHNICAL FIELD

This disclosure relates in general to software build extensibility and in particular, by way of example but not limitation, to a pluggable build architecture that is capable of compiling into an assembly multiple files of arbitrary and/or expandable types.

BACKGROUND

Software, whether executing on a general-purpose processor or a specialized processor, significantly impacts many facets of modern life. When software is to be executed expeditiously, it is often supplied to processors in machine code form. However, it is usually far more difficult and time consuming for human programmers to write software in machine code form as compared to a higher-level language such as Fortran, C++, C#, Visual Basic, and so forth.

Consequently, most software is written in a so-called high-level language and then converted, or compiled, into machine code form. The conversion is effectuated with another software program called a compiler. Compilers take one or more files of a single file type and compile them into a program that is in machine code form.

As software proliferates, the number of different types of files increases. Furthermore, the level of actual and expected interoperability, as well as interconnectedness, between and among various hardware environments and software scenarios likewise continues to grow. Accordingly, there is a need for

1 schemes and/or techniques that can handle different file types as both hardware
2 environments and software scenarios change, grow, and evolve.

3 4 **SUMMARY**

5 In a first exemplary media implementation, one or more processor-
6 accessible media include a build provider that is tailored for a particular file type,
7 the build provider adapted to generate code from files corresponding to the
8 particular file type and to contribute the generated code to a compilation.

9 In a second exemplary media implementation, one or more processor-
10 accessible media include processor-executable instructions that, when executed,
11 direct a device to perform actions including: accepting multiple files, each file of
12 the multiple files corresponding to a respective file type and including source
13 code; associating a build provider with each file of the multiple files in accordance
14 with the corresponding respective file type; ascertaining the source code of each
15 file of the multiple files via the associated build provider; and compiling the
16 ascertained source code of each file of the multiple files into an assembly.

17 Other method, system, approach, apparatus, device, media, application
18 programming interface (API), procedure, arrangement, etc. implementations are
19 described herein.

20 21 **BRIEF DESCRIPTION OF THE DRAWINGS**

22 The same numbers are used throughout the drawings to reference like
23 and/or corresponding aspects, features, and components.

24 FIG. 1 illustrates an exemplary compilation of files having different file
25 types into an assembly using software.

1 FIG. 2 illustrates an exemplary implementation of the software of FIG. 1
2 along with files having different file types.

3 FIG. 3 is a flow diagram that illustrates an exemplary general method for
4 compiling files having different file types into an assembly.

5 FIG. 4 is an exemplary build provider host that illustrates multiple available
6 interfaces thereof.

7 FIG. 5 is an exemplary build provider that illustrates multiple available
8 interfaces thereof.

9 FIG. 6 is a flow diagram that illustrates an exemplary method for compiling
10 files having different file types into an assembly from the perspective of a build
11 provider host and build provider manager.

12 FIG. 7 is a flow diagram that illustrates an exemplary method for compiling
13 files having different file types into an assembly from the perspective of a build
14 provider.

15 FIG. 8 is a block diagram that illustrates an exemplary approach for
16 compiling files having different file types into an assembly.

17 FIG. 9 is an exemplary mapping data structure for build provider
18 registration as shown in FIG. 8.

19 FIG. 10 illustrates an exemplary computing (or general device) operating
20 environment that is capable of (wholly or partially) implementing at least one
21 aspect of software build extensibility as described herein.

22
23 **DETAILED DESCRIPTION**

24 FIG. 1 illustrates an exemplary compilation 110 of files 104 having
25 different file types 108 into an assembly 112 using software 102. Software 102

1 enables multiple files 104 of arbitrary (and possibly expanded/extended) types 108
2 to be compiled 110 into at least one assembly 112. Software 102 may be a
3 primary or a secondary part of a larger program (e.g., an operating system (OS)),
4 or software 102 may be an individual application.

5 As illustrated, three files 104(1), 104(2), and 104(3) are compiled. File 1
6 104(1) includes code 1 106(1) and is of a type A 108A. File 2 104(2) includes
7 code 2 106(2) and is of a type B 108B. File 3 104(3) includes code 3 106(3) and is
8 of a type E 108E. It should be understood that each file 104 may not physically
9 include its code 106. However, the source code for each code 106 is inferable or
10 otherwise derivable from the contents of its file 104. Although a finite number of
11 files 104 and types 108 are illustrated in and/or indicated by FIG. 1, any number of
12 files 104 and types 108 may be involved in a compilation 110 as orchestrated by
13 software 102.

14 File 3 104(3) of type E 108E is shown with dashed lines to indicate that it
15 represents an extended file type. In other words, file 3 104(3) of type E 108E may
16 be compiled 110 into assembly 112 under the control and/or management of
17 software 102 even if software 102 is originally designed and currently exists
18 without direct and/or specific knowledge of files 104 of type E 108E. In fact, files
19 104 of type E 108E may be developed after software 102 is developed.

20 In a described implementation, software 102 provides management and/or
21 hosting as part of an extensible build architecture. In operation, code 1 106(1)
22 from file 1 104(1) of type A 108A, code 2 106(2) from file 2 104(2) of type B
23 108B; and code 3 106(3) from file 3 104(3) of type E 108E are jointly compiled
24 110 into an assembly 112. For example, software 102 may comprise a pluggable
25 build architecture that interfaces with modules assigned to files 104. These

1 modules may be tailored to the corresponding arbitrary file types 108 of files 104
2 in order to facilitate a compilation 110 of their code 106 into a target assembly
3 112.

4 FIG. 2 illustrates an exemplary implementation of the software of FIG. 1
5 along with files 104 having different file types 108. In addition to file 1 104(1),
6 file 2 104(2), and file 3 104(3), a file 4 104(4) that includes code 4 106(4) and is of
7 type A 108A is also being compiled. As illustrated, software 102 (not explicitly
8 indicated in FIG. 2) includes at least one build provider manager 202, one or more
9 build provider hosts 204, and build provider interfaces 206. Additionally, build
10 providers 208 are associated with files 104.

11 In a described implementation, build provider interfaces 206 represent
12 interfaces (e.g., APIs, including methods and/or properties) for (i) build providers
13 208 and (ii) build provider manager 202 and/or build provider host 204.
14 Exemplary build provider interfaces 206 are described below especially for build
15 provider host 204 and build providers 208 with reference to FIGS. 4 and 5,
16 respectively.

17 Build provider manager 202 at least partially manages and/or controls
18 compilation 110 directly and indirectly, including by way of build providers 208
19 and build provider host 204. Build provider manager 202 comprises at least part
20 of a build system. For example, Active Server Pages (ASP) .NET from
21 Microsoft® Corporation of Redmond, Washington includes a general build system.
22 Hence, build provider managers 202 may be implemented for just-in-time (JIT)
23 compiling in a runtime environment, including non-Microsoft® common language
24 runtimes such as the Java™ programming environment from Sun Microsystems®.

1 Nevertheless, build provider managers 202 may alternatively embrace static
2 compilation approaches and other processing environments.

3 Build provider host 204 is implemented by build provider manager 202. A
4 build provider host 204 is typically instantiated once for each compilation 110.
5 Each build provider host 204 provides services to build providers 208 that are
6 involved in the corresponding compilation 110.

7 A respective build provider 208 is created (e.g., instantiated) for and/or
8 assigned to each respective file 104. Build providers 208 are tailored for and/or
9 correspond to particular file types 108. As illustrated, four build providers 208(1),
10 208(2), 208(3), and 208(4) are “plugged into” or interfacing with build provider
11 manager 202. Build provider 208(1) corresponds to type A 108A, build provider
12 208(2) corresponds to type B 108B, build provider 208(3) corresponds to type E
13 108E, and build provider 208(4) also corresponds to type A 108A.

14 Although file 1 104(1) and file 4 104(4) are both of file type A 108A, each
15 is assigned its own build provider 208. Hence, as indicated by the double-empty-
16 headed arrows, build provider 208(1) is associated with file 1 104(1), build
17 provider 208(2) is associated with file 2 104(2), build provider 208(3) is
18 associated with file 3 104(3), and build provider 208(4) is associated with file 4
19 104(4). Although only four files 104 and four associated build providers 208 are
20 illustrated in FIG. 2, any number of such file 104 and build provider 208
21 associations may alternatively participate in a given compilation 110.
22 Furthermore, any number of files 104 of extended type E 108E may be involved.

23 The exemplary extensible build architecture as illustrated in FIG. 2 provides
24 extensibility for new file types 108. A described pluggable build architecture
25 implementation comprises software 102 that enables additions for new build

1 providers 208 that can be associated with the new file types 108. Such software
2 102 enables the addition of new build providers 208 via build provider interfaces
3 206 of build provider host 204 and/or build provider manager 202.

4 Consequently, subsequent and/or outside developers can enable files 104 of
5 extended file types 108E to be compiled 110 by build provider manager 202 along
6 with files 104 of previous file types 108A, 108B, etc. With a relatively minor
7 amount of coding to produce an appropriate build provider 208 for extended file
8 type 108E, developers can enable files of extended file type 108E to be compiled.
9 Moreover, developers can do so without having to worry about the complicated
10 details surrounding compilation 110 and the resulting assembly 112. For example,
11 build provider manager 202 and/or a more-encompassing (e.g., runtime)
12 component can handle where the resulting assembly 112 should and is to live, how
13 assembly 112 can be cached to disk so that it need not be recompiled every time it
14 is to be used, and so forth.

15 FIG. 3 is a flow diagram 300 that illustrates an exemplary general method
16 for compiling files having different file types into an assembly. Flow diagram 300
17 includes four blocks 302-308. Although the actions of flow diagram 300 may be
18 performed in other environments and with a variety of e.g. software schemes,
19 FIGS. 1 and 2 are used in particular to illustrate certain aspects and examples of
20 the method. For example, the actions of blocks 302-308 may be performed by the
21 exemplary extensible build architecture of FIG. 2 in conjunction with exemplary
22 compilation 110 of FIG. 1.

23 At block 302, multiple files of different, arbitrary types are accepted. For
24 example, build provider manager 202 may accept for compilation file 1 104(1),
25 file 2 104(2), file 3 104(3), and file 4 104(4). The files may be of one, two, three,

1 or more different file types 108. For each particular file type 108, there may be
2 one, two, three, or more different files 104 of that particular file type 108. The
3 different, arbitrary file types 108 may possibly include a new expanded file type
4 108E.

5 Examples of types 108 include “.cs” (C# or C sharp), “.aspx” (pages),
6 “.ascx” (user controls), “.asmx” (web services), “.ashx” (web handlers), “.wsdl”
7 (web server description language file), “.arb” (arbitrary type), “.ext” (new
8 extended type), and so forth. Although the preceding exemplary file types are
9 indicated by file extension, other implementations may utilize an alternative
10 indication scheme. For example, the file naming schemes for an Apple® OS, a
11 Unix® OS, a Linux® OS, etc. may alternatively be used to indicate file types 108.
12 Furthermore, file types 108 may be indicated in manners other than a file-naming
13 scheme (e.g., a different file type attribute, tag, etc.).

14 At block 304, a build provider is associated with each respective file of the
15 multiple files according to its corresponding file type. For example, build
16 providers 208(1), 208(2), 208(3), and 208(4) may be associated with files 1
17 104(1), 2 104(2), 3 104(3), and 4 104(4), respectively. A one-to-one
18 correspondence may be established in certain implementations between files 104
19 and build providers 208 such that multiple build providers 208 corresponding to a
20 single file type 108 are instantiated when multiple files 104 of the single file type
21 108 are accepted. For instance, file 1 104(1) and file 4 104(4) are both of type A
22 108A, so two build providers 208 that are tailored for type A 108A are instantiated
23 (e.g., build provider 208(1) and build provider 208(4)).

24 At block 306, source code for each of the multiple files is ascertained via
25 the associated build providers. For example, respective build providers 208 may

ascertain the code 106 of respective associated files 104. For instance, build provider 208(1) ascertains code 1 106(1) from file 1 104(1), build provider 208(2) ascertains code 2 106(2) from file 2 104(2), build provider 208(3) ascertains code 3 106(3) from file 3 104(3), and build provider 208(4) ascertains code 4 106(4) from file 4 104(4).

At block 308, the ascertained source code of the multiple files is compiled into an assembly. For example, build provider host 204 (and/or build provider manager 202) may cause code 1 106(1), code 2 106(2), code 3 106(3), and code 4 106(4) to be compiled 110 into assembly 112. Assembly 112 may be, for example, machine-consumable object code, a dynamic link library or executable file in any general OS environment (e.g., a .dll file or a .exe file, respectively, in a Microsoft® Windows environment), intermediate language (IL) code that is subsequently JIT compiled in a runtime environment (e.g., a common language runtime (CLR) from any given vendor), some combination thereof, and so forth.

FIG. 4 is an exemplary build provider host 204 that illustrates multiple available interfaces 402-414 thereof. Specifically, build provider host 204 includes the following exemplary methods and properties: get referenced assemblies 402, add assembly reference 404, create code file object 406, get code file path 408, get code object model provider 410, add code compile unit 412, and create embedded resource 414. Although seven exemplary interfaces 402-414 are shown and described with respect to build provider host 204, alternative implementations may have more or fewer such interfaces.

In a described implementation, interfaces 402-414 facilitate actions and/or communications between build provider host 204 and multiple build providers 208, especially with regard to providing services to build providers 208 during

1 compilation 110. Get referenced assemblies 402 returns a collection of one or
2 more assemblies with which build providers 208 are intended to be compiled. Add
3 assembly reference 404 adds at least one assembly that can be referenced during
4 compilation 110. For example, if a particular build provider 208 needs or prefers a
5 given assembly in order to have the particular code 106 of its particular file 104
6 compiled, then that particular build provider 208 requests that the given assembly
7 be included in compilation 110.

8 Create code file object 406 creates a file object that is to include new source
9 code for compilation 110. A build provider 208 adds new source code 106 to the
10 file object from an associated file 104. The new source code can subsequently be
11 included in compilation 110 from the code file object. An example of a suitable
12 code file object mechanism is TextWriter of Microsoft® Corporation's ASP .NET.
13 With a TextWriter implementation, build provider 208 writes the new source code
14 to a file using the returned TextWriter. Another example of a suitable code file
15 object mechanism is StringWriter of Java™ from Sun Microsystems®.

16 Get code file path 408 returns a path to a file whose source code is to be
17 included in compilation 110. In a described implementation, the path is a physical
18 file path; however, the path may alternatively employ a virtual or some other
19 mechanism. Additionally, the file is typically actually created by build provider
20 208, instead of build provider host 204. After creation of the returned file, build
21 provider 208 adds to it the new source code 106 of an associated file 104 for
22 compilation 110. In an environment that utilizes file extensions, the source file is
23 given the correct extension for the designated language, as is addressed further
24 herein below.

1 Get code object model provider 410 returns a code object model provider
2 that can be used by build provider 208 to generate a code compile unit. A code
3 compile unit is a high-level, language independent expression of a coding
4 construct. The code object model provider is a mechanism for describing the
5 coding in a language independent manner as a code compile unit. An example of
6 such a code object model provider/code compile unit paradigm is the CodeDOM
7 aspect of Microsoft® Corporation's ASP .NET. With CodeDOM, the employed
8 mechanism for expressing the desired coding construct is an object tree structure.

9 Add code compile unit 412 enables a build provider 208 to add a code
10 compile unit to a compilation 110. Add code compile unit 412 is typically used
11 after get code object model provider 410 in conjunction with a code compile unit
12 that has been generated by the acquired code object model provider. Add code
13 compile unit 412 may therefore be used in lieu of create code file object 406 or get
14 code file path 408 by a build provider 208 that is attempting to contribute its code
15 106 of its associated file 104 to compilation 110 in a language-independent
16 manner. A code object model provider/code compile unit paradigm is described
17 further below with reference to FIGS. 6 and 7, especially with regard to blocks
18 610, 614, and 710”.

19 Create embedded resource 414 creates a new resource that is to be added to
20 compilation 110. The calling build provider 208 can write the desired resource
21 using a returned stream, for example. Examples of such resources that build
22 providers 208 may wish to include in compilation 110 are: localizable resources,
23 text localized to different languages, an image file, and so forth.

24 FIG. 5 is an exemplary build provider 208 that illustrates multiple available
25 interfaces 502-506 thereof. Specifically, build provider 208 includes the following

1 exemplary methods and properties: usable code language 502, generate code 504,
2 and file path 506. Although three exemplary interfaces 502-506 are shown and
3 described with respect to build provider 208, alternative implementations may
4 have more or fewer such interfaces.

5 In a described implementation, interfaces 502-506 facilitate actions and/or
6 communications between build providers 208 and build provider host 204,
7 especially with regard to participation by build providers 208 in compilation 110.
8 Usable code language 502 returns a language that build provider 208 uses, or it
9 can return null if build provider 208 can use any language (i.e., if build provider
10 208 is language agnostic).

11 Generate code 504 asks build provider 208 to generate code 106 of a file
12 104 to which it is associated. The generation/contribution can be effectuated using
13 any of a variety of mechanisms, including those mechanisms (e.g., methods)
14 exposed by build provider host 204. These mechanisms include (i) create code file
15 object 406, (ii) get code file path 408, (iii) get code object model provider 410/add
16 code compile unit 412, (iv) some combination thereof, and so forth.

17 File path 506 gets or sets a path to the associated file 104 that build
18 provider 208 is responsible for handling during compilation 110. In a described
19 implementation, the path is a virtual file path; however, the path may alternatively
20 be physical or employ some other file identification mechanism. For a virtual path
21 implementation, a virtual path may map to a physical path, without necessarily
22 having a one-to-one correspondence between the two different kinds of paths.

23 FIGS. 6-9 are described together with interrelated references to each other.
24 FIG. 6 is a flow diagram 600 that illustrates an exemplary method for compiling
25 files having different file types into an assembly from the perspective of a build

1 provider host 204 and build provider manager 202. Flow diagram 600 includes
2 seven blocks 602-614. Although the actions of flow diagram 600 may be
3 performed in other environments and with a variety of e.g. software schemes,
4 FIGS. 1, 2, 4, and 5 and in particular 8 and 9 are used to illustrate certain aspects
5 and examples of the method.

6 FIG. 7 is a flow diagram 700 that illustrates an exemplary method for
7 compiling files having different file types into an assembly from the perspective of
8 a build provider 208. Flow diagram 700 includes seven blocks 704, 706, 708,
9 710', 710'', 710''', and 712. Although the actions of flow diagram 700 may be
10 performed in other environments and with a variety of e.g. software schemes,
11 FIGS. 1, 2, 4, and 5 and in particular 8 and 9 are used to illustrate certain aspects
12 and examples of the method. Generally, respective actions 604-612 of FIG. 6 are
13 related to respective actions 704-712 of FIG. 7.

14 FIG. 8 is a block diagram 800 that illustrates an exemplary approach for
15 compiling files 104 having different file types into an assembly 112. FIG. 9 is an
16 exemplary mapping data structure 802 for build provider registration as shown in
17 FIG. 8. Block diagram 800 includes a build provider manager 202 and a build
18 provider host 204 thereof. Build provider manager 202 has access to build
19 provider registration – mapping data structure (BPR – MDS) 802.

20 As illustrated, three files 104 include code 106. Specifically, file 1 104(1)
21 includes code 1 106(1), file 2 104(2) includes code 2 106(2), and file 3 104(3)
22 includes code 3 106(3). Also, three respective files 104 are associated with three
23 respective build providers 208. Specifically, file 1 104(1) is associated with build
24 provider 208(1), file 2 104(2) is associated with build provider 208(2), and file 3
25 104(3) is associated with build provider 208(3). Although not explicitly shown in

1 FIG. 8 for the sake of clarity, each file 104 and each associated build provider 208
2 correspond to a particular file type 108 (e.g., as shown in FIG. 2).

3 In a described implementation, build provider manager 202 and/or build
4 provider host 204 manage and/or control compilation 110. Compilation 110
5 entails compiling the source code of code 1 106(1), code 2 106(2), and code 3
6 106(3) into a single target assembly 112. The files 104, in which the source code
7 of code 106 is located, may be of arbitrary and different file formats that are
8 possibly unrelated to each other.

9 For flowchart 600 (of FIG. 6) at block 602, multiple files are accepted. For
10 example, build provider manager 202 may accept file 1 104(1), file 2 104(2), and
11 file 3 104(3). As shown in the example of FIG. 2, file 1 104(1) corresponds to type
12 A 108A, file 2 104(2) corresponds to type B 108B, file 3 104(3) corresponds to
13 type E 108E.

14 At block 604, an associated build provider is created for each file. For
15 example, with reference to BPR – MDS 802, build provider manager 202 may
16 create a respective build provider 208 for each respective file 104. For instance,
17 build provider 208(1) is created for file 1 104(1), build provider 208(2) is created
18 for file 2 104(2), and build provider 208(3) is created for file 3 104(3). In a
19 described implementation, BPR – MDS 802 maps file types 108 to different types
20 of build providers 208.

21 In FIG. 9, BPR – MDS 802 has multiple entries 902 in which each entry
22 902 includes a particular file type 108 and a denotation of a build provider 208
23 that can handle files 104 of that particular file type 108. Specifically, entry 902(1)
24 maps file type 108(1) to (a denotation of) build provider 208(1), entry 902(2) maps
25 file type 108(2) to (a denotation of) build provider 208(2), and entry 902(n) maps

1 file type 108(n) to (a denotation of) build provider 208(n). As indicated by the
2 index “n”, any number of mapping entries 902 may be included in BPR – MDS
3 802.

4 In certain implementations, such as those in a Microsoft® Windows®
5 environment, BPR – MDS 802 may be realized as a registration portion of a
6 configuration file. Furthermore, each file type 108 may be indicated by a file
7 extension such as .cs, .aspx, .ascx, .asmx, .ashx, .wsdl, .arb, “.new/.abc” (for a new
8 type), .ext, and so forth.

9 Hence, build providers 208 that are to participate in compilations 110 are
10 registered in a configuration file. Although a BPR – MDS 802 may be utilized in
11 other OS environments, an example of a BPR – MDS 802 as part of a
12 configuration file in a Microsoft® Windows® environment follows:

```
13 <buildProviders>  
14 <add extension=".acme" type="Acme.MyCustomBuildProvider,  
15 AcmeAssembly" />  
16 </buildProviders>
```

17 This registers a build provider 208 to handle files with an “.acme” extension by
18 mapping a file type 108 of “.acme” to build provider 208 of
19 “Acme.MyCustomBuildProvider”. This build provider 208 lives in the class
20 Acme.MyCustomBuildProvider in the assembly AcmeAssembly.dll. The build
21 provider 208 of “Acme.MyCustomBuildProvider” extends an exemplary
22 “BuildProvider” class, which is described below in a section entitled “Exemplary
23 Descriptions for BuildProviderHost and BuildProvider Classes”.

24 For flowchart 700 (of FIG. 7) at block 704, a build provider is created. As
25 indicated by the rounded rectangle 704 and the dashed arrow extending therefrom,

1 build provider 208 is created (e.g., instantiated) prior to the actions of blocks 706-
2 712 being performed by build provider 208.

3 At block 606, each build provider is given a path to its associated file. For
4 example a path (e.g., a physical or virtual path) for file 1 104(1) is given to build
5 provider 208(1), a path for file 2 104(2) is given to build provider 208(2), and a
6 path for file 3 104(3) is given to build provider 208(3). These paths may be given
7 by build provider manager 202 and/or build provider host 204 to build providers
8 208(1, 2, and 3) by calling their respective file path 506 interfaces. At block 706,
9 each build provider receives the path of its associated file. For example,
10 respective build providers 208(1, 2, and 3) receive paths for their respective files
11 104(1, 2, and 3) via their respective file path 506 interfaces.

12 At block 608, each build provider is asked for its usable language.* For
13 example, build provider manager 202 and/or build provider host 204 invoke the
14 usable code language 502 method of each build provider 208(1, 2, and 3). In
15 response, at block 708, each build provider indicates its usable code language.
16 Each build provider 208 can use a specific language for the source code 106 of its
17 associated file 104, or it can use any language. Thus, each build provider 208 can
18 indicate a specific language (e.g., C#, Visual Basic, etc.) or that it does not care
19 which language is used (e.g., that it is language agnostic by returning null). The
20 language indicated by build providers 208 is designated as the language that is to
21 be used for compilation 110.

22 In a described implementation, in order to compile 110 the various codes
23 106 into one assembly 112, each build provider 208 has the same language or is
24 language agnostic. In other words, assembly 112 is formed from codes 106 that
25 are from the same language or are language independent (e.g., under a get code

1 object model provider 410/add code compile unit 412 mechanism). Thus, for a
2 group of codes 106 that is to be compiled 110 into an assembly 112, each code 106
3 of the group is all the same language, all language agnostic, or all the same
4 language with some that are language agnostic. If all build providers 208 for a
5 given grouping are language agnostic, then a default language is used as the
6 designated language.

7 At some time after respective build providers 208 are associated with
8 respective files 104 (at blocks 604 and 704), build providers 208 may optionally
9 call the get referenced assemblies 402 interface of build provider host 204. The
10 assemblies to be referenced in compilation 110 are returned to build providers 208.
11 Based on each build provider's 208 associated file 104, for example, each build
12 provider 208 determines whether an additional assembly or assemblies may be
13 required or preferred when compiling the code 106 included as part of its
14 associated file 104. If so, the relevant build providers 208 call the add assembly
15 reference 404 interface of build provider host 204 to have such assembly or
16 assemblies added. Of course, a particular assembly may be added once by build
17 provider host 204 regardless of the number of relevant build providers 208 that
18 call add assembly reference 404 for the particular assembly. This add assembly
19 reference 404 interface can be especially helpful when classes are being extended
20 in a given compilation 110.

21 At block 610, each build provider is requested to contribute code. For
22 example, build provider host 204 may make a call to the generate code 504
23 interface of each build provider 208(1), 208(2), and 208(3) (e.g., in a sequential
24 fashion). As indicated by blocks 710', 710'', and 710''', build providers 208 may
25 contribute code 106 of their respective files 104 in any of three different

1 exemplary manners/mechanisms for compilation 110. It should be noted that a
2 given build provider 208 can contribute code 106 more than once, either by using
3 the same contribution manner multiple times or by using an arbitrary combination
4 of any two or more of the three described (or other) contribution manners. The
5 three contribution manners of a described implementation are shown at blocks
6 804, 806, and 808 of FIG. 8. As described further below, block 804 relates to
7 block 710', block 806 relates to block 710'', and block 808 relates to block 710'''.

8 Generally, each build provider 208 is associated with a file 104 of a type
9 108 to which each corresponds. Consequently, a particular build provider 208 of a
10 particular type 108 is adapted to comprehend, parse, dissect, etc. an associated
11 particular file 104 of the particular type 108, and it is therefore capable of
12 generating the code 106 from the associated particular file 104. It should be noted
13 that code 106 may be contiguous or discontinuous, comprised of one or more
14 modules, intermixed with other non-code portions, directly or indirectly derived
15 from non-code portion(s), and so forth.

16 Depending on the format of the particular type 108 and/or the
17 capabilities/configuration of the particular build provider 208, the generated code
18 106 that is to be contributed to compilation 110 may or may not be the entirety of
19 the code that is included as part of the particular file 104. For example, a section
20 of code may be commented out, a particular code portion may not be applicable to
21 compilation 110 and/or the target assembly 112, and so forth. Furthermore, build
22 providers 208 may have the ability to generate (and therefore contribute) source
23 code from non-code portion(s) of files 104. Thus, source code to be contributed to
24 a compilation 110 is generated based on a particular build provider 208 and a
25

1 particular associated file 104 (or more generally from files 104 of the same
2 corresponding type 108 as the particular build provider 208).

3 At block 710', source code is written to an object. For example, build
4 provider 208(1) may cause code 1 106(1) of file 1 104(1) to be written to a code
5 file object at block 804 in order to contribute code 1 106(1) to compilation 110.
6 For instance, build provider 208(1) may call the create code file object 406
7 interface of build provider host 204 in order to acquire a code file object to which
8 code 1 106(1) may be added.

9 At block 710'', source code is written to a stipulated path. For example,
10 build provider 208(2) may write code 2 106(2) of file 2 104(2) to a file at a
11 location stipulated by a path acquired from build provider host 204 at block 806 in
12 order to contribute code 2 106(2) to compilation 110. For instance, build provider
13 208(2) may call the get code file path 408 interface of build provider host 204 in
14 order to acquire the path to a file to be created by build provider 208(2) to which
15 code 2 106(2) is added.

16 At block 710''', a code object model (COM) provider is requested and the
17 code object model provider is used to generate a code compile unit. For example,
18 build provider 208(3) may request a code object model provider from build
19 provider host 204, and build provider 208(3) may subsequently employ the code
20 object model provider to generate a code compile unit for code 3 106(3) of file 3
21 104(3) at block 808 in order to contribute code 3 106(3) to compilation 110. For
22 instance, build provider 208(3) may call the get code object model provider 410
23 interface as well as the add code compile unit 412 interface of build provider host
24 204 in order to acquire and use a code object model provider.

1 The code compile unit 808 for code 3 106(3) is generated, and may be
2 contributed, as a language-independent structure. In this example, build provider
3 208(3) is language agnostic inasmuch as the language-independent structure may
4 be converted into any desired language. In a described implementation, build
5 provider host 204 performs or causes to be performed the conversion of the
6 language-independent structure to source code in the designated language for
7 compilation 110. Alternatively, some other entity, such as build provider manager
8 202 or build provider 208(3), may perform this conversion.

9 As illustrated in FIG. 8, build provider 208(1) uses a writable object
10 mechanism for code contribution, build provider 208(2) uses a writable file path
11 location mechanism for code contribution, and build provider 208(3) uses a code
12 object model provider/code compile unit mechanism for code contribution.
13 However, any given build provider 208 may be capable of using any one or more
14 mechanisms for contributing code, optionally including the use of multiple
15 mechanisms in a single compilation 110.

16 By way of example, new file types 108E that correspond to an extended
17 build provider 208 (e.g., build provider 208(3)) may use a writable object
18 mechanism and/or a writable file path mechanism, as well as the illustrated code
19 object model provider/code compile unit mechanism. Likewise, a pre-planned or
20 built-in build provider 208 (e.g., build providers 208(1) and 208(2)) may employ a
21 code object model provider/code compile unit mechanism. Furthermore, build
22 providers 208 that are not language-agnostic may use a code object model/code
23 compile unit mechanism. In such cases, the code object model contains some
24 nodes that are language-specific along with other nodes that are language-
25 agnostic.

1 At block 712, zero, one or more resources are submitted. For example,
2 build providers 208(1, 2, and/or 3) may submit to build provider host 204
3 resource(s) for use in compilation 110. For instance, each relevant build provider
4 208 may call the create embedded resource 414 interface of build provider host
5 204. At block 612, resources (including notifications thereof) are received from
6 build providers 208. For example, build provider host 204 may receive submitted
7 resources (not explicitly shown in FIG. 8) from one or more build providers 208.

8 At block 614, code and resources (if any) from multiple build providers are
9 compiled into an assembly. For example, build provider host 204 may launch a
10 compiler (e.g., a compiler for the designated language) to compile code 1 106(1)
11 from the writable object of block 804, code 2 106(2) from the stipulated file
12 location of block 806, and code 3 106(3) from the language-converted code
13 compile unit of block 808 into target assembly 112. Compilation 110 therefore
14 causes assembly 112 to include and be derived from source code of code 1 106(1),
15 code 2 106(2), and code 3 106(3). This compilation 110 may be effectuated even
16 when a file type 108 of a file 104 that is participating in compilation 110 is
17 developed after build provider manager 202 and/or build provider host 204 is
18 developed, as well as when files 104 are unrelated to one another.

19 The actions, aspects, features, components, etc. of FIGS. 1-9 are illustrated
20 in diagrams that are divided into multiple blocks. However, the order,
21 interconnections, interrelationships, layout, etc. in which FIGS. 1-9 are described
22 and/or shown is not intended to be construed as a limitation, and any number of
23 the blocks can be modified, combined, rearranged, augmented, omitted, etc. in any
24 manner to implement one or more systems, methods, devices, procedures, media,
25 APIs, apparatuses, arrangements, etc. for software build extensibility.

1 Furthermore, although the description herein includes references to specific
2 implementations (and the exemplary operating environment of FIG. 10), the
3 illustrated and/or described implementations can be implemented in any suitable
4 hardware, software, firmware, or combination thereof and using any suitable
5 software architecture(s), source code language(s), code contribution
6 mechanism(s), compiling scheme(s), and so forth.

7 **Exemplary Operating Environment for Computer or Other Device**

8 FIG. 10 illustrates an exemplary computing (or general device) operating
9 environment 1000 that is capable of (fully or partially) implementing at least one
10 system, device, apparatus, component, arrangement, protocol, approach, method,
11 procedure, media, API, some combination thereof, etc. for software build
12 extensibility as described herein. Operating environment 1000 may be utilized in
13 the computer and network architectures described below.

14 Exemplary operating environment 1000 is only one example of an
15 environment and is not intended to suggest any limitation as to the scope of use or
16 functionality of the applicable device (including computer, network node,
17 entertainment device, mobile appliance, general electronic device, etc.)
18 architectures. Neither should operating environment 1000 (or the devices thereof)
19 be interpreted as having any dependency or requirement relating to any one or to
20 any combination of components as illustrated in FIG. 10.

21 Additionally, software build extensibility may be implemented with
22 numerous other general purpose or special purpose device (including computing
23 system) environments or configurations. Examples of well known devices,
24 systems, environments, and/or configurations that may be suitable for use include,
25 but are not limited to, personal computers, server computers, thin clients, thick

1 clients, personal digital assistants (PDAs) or mobile telephones, watches, hand-
2 held or laptop devices, multiprocessor systems, microprocessor-based systems,
3 set-top boxes, programmable consumer electronics, video game machines, game
4 consoles, portable or handheld gaming units, network PCs, minicomputers,
5 mainframe computers, network nodes, distributed or multi-processing computing
6 environments that include any of the above systems or devices, some combination
7 thereof, and so forth.

8 Implementations for software build extensibility may be described in the
9 general context of processor-executable instructions. Generally, processor-
10 executable instructions include routines, programs, modules, protocols, objects,
11 interfaces, components, data structures, etc. that perform and/or enable particular
12 tasks and/or implement particular abstract data types. Software build extensibility,
13 as described in certain implementations herein, may also be practiced in
14 distributed processing environments where tasks are performed by remotely-linked
15 processing devices that are connected through a communications link and/or
16 network. Especially but not exclusively in a distributed computing environment,
17 processor-executable instructions may be located in separate storage media,
18 executed by different processors, and/or propagated over transmission media.

19 Exemplary operating environment 1000 includes a general-purpose
20 computing device in the form of a computer 1002, which may comprise any (e.g.,
21 electronic) device with computing/processing capabilities. The components of
22 computer 1002 may include, but are not limited to, one or more processors or
23 processing units 1004, a system memory 1006, and a system bus 1008 that couples
24 various system components including processor 1004 to system memory 1006.

1 Processors 1004 are not limited by the materials from which they are
2 formed or the processing mechanisms employed therein. For example, processors
3 1004 may be comprised of semiconductor(s) and/or transistors (e.g., electronic
4 integrated circuits (ICs)). In such a context, processor-executable instructions may
5 be electronically-executable instructions. Alternatively, the mechanisms of or for
6 processors 1004, and thus of or for computer 1002, may include, but are not
7 limited to, quantum computing, optical computing, mechanical computing (e.g.,
8 using nanotechnology), and so forth.

9 System bus 1008 represents one or more of any of many types of wired or
10 wireless bus structures, including a memory bus or memory controller, a point-to-
11 point connection, a switching fabric, a peripheral bus, an accelerated graphics port,
12 and a processor or local bus using any of a variety of bus architectures. By way of
13 example, such architectures may include an Industry Standard Architecture (ISA)
14 bus, a Micro Channel Architecture (MCA) bus, an Enhanced ISA (EISA) bus, a
15 Video Electronics Standards Association (VESA) local bus, a Peripheral
16 Component Interconnects (PCI) bus also known as a Mezzanine bus, some
17 combination thereof, and so forth.

18 Computer 1002 typically includes a variety of processor-accessible media.
19 Such media may be any available media that is accessible by computer 1002 or
20 another (e.g., electronic) device, and it includes both volatile and non-volatile
21 media, removable and non-removable media, and storage and transmission media.

22 System memory 1006 includes processor-accessible storage media in the
23 form of volatile memory, such as random access memory (RAM) 1040, and/or
24 non-volatile memory, such as read only memory (ROM) 1012. A basic
25 input/output system (BIOS) 1014, containing the basic routines that help to

1 transfer information between elements within computer 1002, such as during start-
2 up, is typically stored in ROM 1012. RAM 1010 typically contains data and/or
3 program modules/instructions that are immediately accessible to and/or being
4 presently operated on by processing unit 1004.

5 Computer 1002 may also include other removable/non-removable and/or
6 volatile/non-volatile storage media. By way of example, FIG. 10 illustrates a hard
7 disk drive or disk drive array 1016 for reading from and writing to a (typically)
8 non-removable, non-volatile magnetic media (not separately shown); a magnetic
9 disk drive 1018 for reading from and writing to a (typically) removable, non-
10 volatile magnetic disk 1020 (e.g., a "floppy disk"); and an optical disk drive 1022
11 for reading from and/or writing to a (typically) removable, non-volatile optical
12 disk 1024 such as a CD, DVD, or other optical media. Hard disk drive 1016,
13 magnetic disk drive 1018, and optical disk drive 1022 are each connected to
14 system bus 1008 by one or more storage media interfaces 1026. Alternatively,
15 hard disk drive 1016, magnetic disk drive 1018, and optical disk drive 1022 may
16 be connected to system bus 1008 by one or more other separate or combined
17 interfaces (not shown).

18 The disk drives and their associated processor-accessible media provide
19 non-volatile storage of processor-executable instructions, such as data structures,
20 program modules, and other data for computer 1002. Although exemplary
21 computer 1002 illustrates a hard disk 1016, a removable magnetic disk 1020, and a
22 removable optical disk 1024, it is to be appreciated that other types of processor-
23 accessible media may store instructions that are accessible by a device, such as
24 magnetic cassettes or other magnetic storage devices, flash memory, compact
25 disks (CDs), digital versatile disks (DVDs) or other optical storage, RAM, ROM,

1 electrically-erasable programmable read-only memories (EEPROM), and so forth.
2 Such media may also include so-called special purpose or hard-wired IC chips. In
3 other words, any processor-accessible media may be utilized to realize the storage
4 media of the exemplary operating environment 1000.

5 Any number of program modules (or other units or sets of
6 instructions/code) may be stored on hard disk 1016, magnetic disk 1020, optical
7 disk 1024, ROM 1012, and/or RAM 1040, including by way of general example,
8 an operating system 1028, one or more application programs 1030, other program
9 modules 1032, and program data 1034.

10 A user may enter commands and/or information into computer 1002 via
11 input devices such as a keyboard 1036 and a pointing device 1038 (e.g., a
12 "mouse"). Other input devices 1040 (not shown specifically) may include a
13 microphone, joystick, game pad, satellite dish, serial port, scanner, and/or the like.
14 These and other input devices are connected to processing unit 1004 via
15 input/output interfaces 1042 that are coupled to system bus 1008. However, input
16 devices and/or output devices may instead be connected by other interface and bus
17 structures, such as a parallel port, a game port, a universal serial bus (USB) port,
18 an infrared port, an IEEE 1394 ("Firewire") interface, an IEEE 802.11 wireless
19 interface, a Bluetooth® wireless interface, and so forth.

20 A monitor/view screen 1044 or other type of display device may also be
21 connected to system bus 1008 via an interface, such as a video adapter 1046.
22 Video adapter 1046 (or another component) may be or may include a graphics
23 card for processing graphics-intensive calculations and for handling demanding
24 display requirements. Typically, a graphics card includes a graphics processing
25 unit (GPU), video RAM (VRAM), etc. to facilitate the expeditious display of

1 graphics and performance of graphics operations. In addition to monitor 1044,
2 other output peripheral devices may include components such as speakers (not
3 shown) and a printer 1048, which may be connected to computer 1002 via
4 input/output interfaces 1042.

5 Computer 1002 may operate in a networked environment using logical
6 connections to one or more remote computers, such as a remote computing device
7 1050. By way of example, remote computing device 1050 may be a personal
8 computer, a portable computer (e.g., laptop computer, tablet computer, PDA,
9 mobile station, etc.), a palm or pocket-sized computer, a watch, a gaming device, a
10 server, a router, a network computer, a peer device, another network node, or
11 another device type as listed above, and so forth. However, remote computing
12 device 1050 is illustrated as a portable computer that may include many or all of
13 the elements and features described herein with respect to computer 1002.

14 Logical connections between computer 1002 and remote computer 1050 are
15 depicted as a local area network (LAN) 1052 and a general wide area network
16 (WAN) 1054. Such networking environments are commonplace in offices,
17 enterprise-wide computer networks, intranets, the Internet, fixed and mobile
18 telephone networks, ad-hoc and infrastructure wireless networks, other wireless
19 networks, gaming networks, some combination thereof, and so forth. Such
20 networks and communications connections are examples of transmission media.

21 When implemented in a LAN networking environment, computer 1002 is
22 usually connected to LAN 1052 via a network interface or adapter 1056. When
23 implemented in a WAN networking environment, computer 1002 typically
24 includes a modem 1058 or other component for establishing communications over
25 WAN 1054. Modem 1058, which may be internal or external to computer 1002,

1 may be connected to system bus 1008 via input/output interfaces 1042 or any
2 other appropriate mechanism(s). It is to be appreciated that the illustrated network
3 connections are exemplary and that other manners for establishing communication
4 link(s) between computers 1002 and 1050 may be employed.

5 In a networked environment, such as that illustrated with operating
6 environment 1000, program modules or other instructions that are depicted
7 relative to computer 1002, or portions thereof, may be fully or partially stored in a
8 remote media storage device. By way of example, remote application programs
9 1060 reside on a memory component of remote computer 1050 but may be usable
10 or otherwise accessible via computer 1002. Also, for purposes of illustration,
11 application programs 1030 and other processor-executable instructions such as
12 operating system 1028 are illustrated herein as discrete blocks, but it is recognized
13 that such programs, components, and other instructions reside at various times in
14 different storage components of computing device 1002 (and/or remote computing
15 device 1050) and are executed by processor(s) 1004 of computer 1002 (and/or
16 those of remote computing device 1050).

17 **Exemplary Descriptions for BuildProviderHost and BuildProvider Classes**

18 Two exemplary classes involved in a described architecture are:
19 BuildProviderHost and BuildProvider. BuildProviderHost may be implemented,
20 for example, by the ASP.NET build system from Microsoft® Corporation.
21 BuildProvider may be implemented for each file type that plugs into the overall
22 build system.

23 An exemplary description of a BuildProvider class follows an exemplary
24 description of a BuildProviderHost class:
25

```

1  /// Provides services to BuildProvider's during their compilation
2  public abstract class BuildProviderHost {
3
4      /// Returns a collection of assemblies that the build provider is to be compiled with.
5      public abstract ICollection GetReferencedAssemblies();
6
7      /// Adds an assembly that is to be referenced during compilation.
8      public abstract void AddAssemblyReference(Assembly a);
9
10     /// Returns a CodeDomProvider that the build provider can use to generate a
11     /// CodeCompileUnit.
12     public abstract CodeDomProvider GetCodeDomProvider();
13
14     /// Creates a new source file that will be added to the compilation. The build
15     /// provider writes source code to this file using the returned TextWriter.
16     /// The build provider may close the TextWriter when it is done writing to it.
17     /// The build provider passes itself as a parameter to this method.
18     public abstract TextWriter CreateCodeFile(BuildProvider buildProvider);
19
20     /// Returns the physical path to a source file that will be included in the
21     /// compilation. Note that the file is not actually created. It is up to the
22     /// build provider to do this.
23     /// The source file has the correct extension for the target language.
24     /// The build provider passes itself as a parameter to this method.
25     public abstract string GetCodeFilePhysicalPath(BuildProvider buildProvider);
26
27     /// Adds a CodeCompileUnit to the compilation. This is typically used as an
28     /// alternative to CreateSourceFile, esp. by providers who are CodeDOM aware.
29     /// The build provider passes itself as a parameter to this method.

```

```
1 public abstract void AddCodeCompileUnit(BuildProvider buildProvider,  
2 CodeCompileUnit codeCompileUnit);
```

```
3     /// Creates a new resource that is to be added to the compilation. The build
```

```
4     /// provider can write to it using the returned Stream.
```

```
5     /// The build provider may close the Stream when it is done writing to it.
```

```
6     /// The build provider passes itself as a parameter to this method.
```

```
7     public abstract Stream CreateEmbeddedResource(BuildProvider buildProvider,  
8     string name);
```

```
9  
10    /// Base class for build providers that want to participate in a compilation.
```

```
11    /// It may be used by build providers that process files based on a virtual path.
```

```
12    public abstract class BuildProvider {
```

```
13        /// Returns the language that this build provider uses, or null if it can use
```

```
14        /// any language.
```

```
15        public virtual string GetCodeLanguage();
```

```
16        /// Asks this build provider to generate any code that it has, using the various
```

```
17        /// methods on the passed in BuildProviderHost.
```

```
18        public virtual void GenerateCode(BuildProviderHost host);
```

```
19  
20        /// Gets or sets the virtual path that this build provider handles.
```

```
21        public string VirtualPath { get; set; }
```

```
22    }
```

23 The above class descriptions are provided by way of example only, for
24 software build extensibility may be implemented in a myriad of other manners as
25

1 described herein. Additionally, although the above classes are implemented in the
2 C# programming language, they may alternatively be implemented in one or more
3 other languages. Furthermore, the above classes may be alternatively
4 implemented in one or more non-Microsoft® Corporation environments.

5 Although systems, media, devices, methods, procedures, apparatuses,
6 techniques, APIs, schemes, approaches, procedures, arrangements, and other
7 implementations have been described in language specific to structural, logical,
8 algorithmic, and functional features and/or diagrams, it is to be understood that the
9 invention defined in the appended claims is not necessarily limited to the specific
10 features or diagrams described. Rather, the specific features and diagrams are
11 disclosed as exemplary forms of implementing the claimed invention.